Formal Authorization Allocation Approaches for Permission-role assignments Using Relational Algebra Operations

Hua Wang Yanchun Zhang Jinli Cao
Department of Maths & Computing
University of Southern Queensland
Toowoomba, QLD 4350, Australia
(wang, zhang, cao)@usq.edu.au

Abstract

In this paper, we develop formal authorization allocation algorithms for permission-role assignments. The formal approaches are based on relational structure, and relational algebra and operations. The process of permission-role assignments is an important issue in role-based access control (RBAC) as it may modify the authorization level or imply high-level confidential information to be derived when roles are changed and request different permissions. There are two types of problems that may arise in permission-role assignments. One is related to authorization granting process. Conflicting permissions may be granted to a role, and as a result, users with the role may have or derive a high level of authority. Another is related to authorization revocation. When a permission is revoked from a role, the role may still have the permission from other roles.

To solve the problems, this paper presents an authorization granting algorithm, and weak revocation and strong revocation algorithms that are based on relational algebra operations. The algorithms can be used to check conflicts and therefore to help allocate permissions without compromising the security in RBAC. We describe how to use the new algorithms with an anonymity scalable payment scheme. Finally, comparisons with other related work are discussed.

Keywords: RBAC, Permission-role assignment, Authorization, Can-assignp, Can-revokep.

1 Introduction

With people's increased consciousness of the need for electronic commerce to protect their private information and to provide security of applications, system administrators are continuing to implement access control mechanisms as well as to retain a critical and complex aspect of security administration. Traditional administrations of access control are mandatory, discretionary and role-based access control. Mandatory access controls (MAC) restrict access to data based on varying degrees of security requirements for information contained in the objects. Information is associated with multi-level security requirements with labels such as TOP SECRET, SECRET, and CONFIDENTIAL [2]. An assigned right cannot be changed and modifications are permitted only to administrators. Users may need to register on a number of different servers of different operating system types, various databases and multiple business applications. Furthermore, an object classification reflects the sensitivity of the information contained in the object, that is, the potential damage that may come from unauthorized disclosure of information. Registration of each user with each facility is needed to control and prevent unauthorized use. How to manage a system with MAC is a challenging task, especially when dealing with the changes on user positions and other access rights. Discretionary access controls (DAC) allow users to grant or revoke access to any authority under their control without the intercession of a system administrator [8]. Access rights to resources are based on the identity of persons and/or groups to which they belong. When the number of users increases, the management is costly. DAC grants authorization or privileges to users directly, authorized statically when they set up an account. Though it is convenient for users to pass on the authorization directly to other users, it brings a serious problem on security. For example, when a user passes on some access controls to another user, it may change the level of access privilege of the second user who may then able to access or derive high level information based on the level of access control gained.

Role-based access control (RBAC) is an alternative system that involves individual users being associated with roles as well as roles being associated with permissions (Each permission is a pair of objects and operations). As such, a role is used to associate users and permissions. A

user in this model is a human being. A role is a job function or job title within the organization associated with authority and responsibility.

Permission is an approval of a particular operation to be performed on one or more objects. As shown in Figure 1, the relationships between users and roles, and between roles and permissions are many-to-many. (i.e. a permission can be associated with one or more roles, and a role can be associated with one or more permissions). The security policy of the organization determines role membership and the allocation of each roles capabilities.

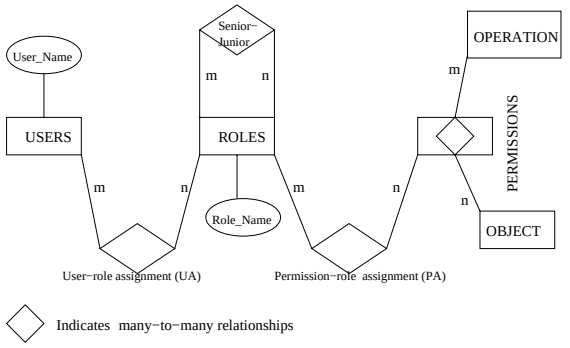

Figure 1. RBAC relationship

There are three advantages of RBAC management. Firstly, it is much easier to manage a system using of RBAC. In RBAC, a security administrator adds transactions to roles or deletes transactions from roles, where transactions can be a program object associated with data [8]. Security issues are addressed by associating programming code and data into a transaction. Access control does not require any checks on the user's or the program's right to access a data item, since the accesses are built into the transaction. Secondly, RBAC can reduce administration cost and complexity [16]. Usually, there is a relationship between the cost of administration and the number of associations which must be managed in order to administer an access control policy. The larger the number of associations, the costlier and more error prone the access control administration is likely to be, but the use of RBAC reduces the number of associations to be managed. Thirdly, RBAC is better than a typical access control list (ACL) model [11]. RBAC can authorize and audit capabilities so that people are simply assigned new roles while they change responsibilities. This allows for the authorities of a person to be easily documented. By contrast, in ACL, the entire set of authorities must be searched to develop a clear picture of a person's rights because ACLs only support the specification of user/permission and group/permission relationships.

Recently, role based access control (RBAC) has been widely used in database system management and operating system products. In 1993, the National Institute of Standards and Technology (NIST) developed prototype implementations, sponsored external research [6], and published formal RBAC models [8, 10]. Many organizations prefer to centrally control and maintain access rights, not so much at the system administrator's personal discretion but more in accordance with the organization's protection guidelines [5]. RBAC is being considered as part of the emerging SQL3 standard for database management systems, based on their implementation in Oracle 7 [15]. Many RBAC practical applications have been implemented [3, 9, 16].

However, there is a consistency problem when using RBAC management. For instance, if there are hundreds of permissions and thousands of roles in a system, it is very difficult to maintain consistency because it may change the authorization level, or imply high-level confidential information to be derived when more than one permission is requested and granted.

We develop formal approaches to check the conflicts and therefore help allocate the permissions without compromising the security. The formal approaches are based on relational structure and relational algebra operations. As far as we know, this is the first kind of work in this area to address the formal approaches for permission allocation and conflict detection.

The paper is organized as follows. In the next section, we identify the problems related to permission assignment and revocation. Relational algebra-based authorization granting algorithm and revocation algorithms are developed in section 3. In section 4, we review an anonymity scalable electronic commerce payment scheme. We then apply the formal authorization approaches to this scheme in section 5. Comparisons with related work are discussed in section 6 and the conclusions are in section 7.

2 Problem Definitions

With RBAC, users cannot associate with permissions directly but roles. Permissions must be authorized for roles, and roles must be authorized for users. The RBAC security model has two components: MC_0 and MC_1 [16]. Model component MC_0, called the RBAC authorization database model, defines the RBAC security properties for authorization of static roles. Static properties of an RBAC authorization database include role hierarchy, inheritance, cardinality, and static separation of duty. MC_1 called the RBAC activation model, defines the RBAC security properties for dynamic activation of roles. Dynamic properties include role activation, permission execution, dynamic separation of duties, and object access. In particular, the RBAC model supports the specification of several aspects.

a. User-role assignmentss – the constraints specifying user authorizations to perform roles;

b. Permission-role assignments – the constraints specifying role authorizations to have permissions;

c. Role hierarchies – the constraints specifying which role may inherit all of the permissions of another role;

d. Duty separation constraints – these are role/role associations indicating conflict of interest:

d1. Static separated duty (SSD) – a constraint specifying that a user cannot be authorized for two different roles;

d2. Dynamic separated duty (DSD) – a constraint specifying that a user can be authorized for two different roles but cannot act simultaneously in both;

e. Cardinality – the maximum number of users allowed, i.e. how many users can be authorized for any particular role (role cardinality), e.g., only one manager.

A comprehensive administrative model for RBAC must account for all issues mentioned above, among others. However, permission-role assignment is a particularly critical administrative activity. This is because conflict defined in terms of roles may allow conflicting permissions to be assigned to the same role but conflicts defined in terms of permissions eliminates this possibility. Therefore, this paper will also focus on permission-role assignments.

Let D be a database with a set of relations REL, a set of attributes A. REL includes ROLES, PERM, ROLE-PERM, SEN-JUN, Can-assignp and Can-revokep etc. A includes attributes such as RoleName, PermName, Senior and Junior etc from the relations. Roles are in two categories, one is administrative roles (admin.role), the other is regular roles (role) that permissions are assigned to or revoked by administrative roles. The permissions assigned to a role by administrators may be in conflicts. For example, the permission for approving loan in a bank and that of funding loan are conflicting. These two permissions cannot be assigned to a role; on the other hand, because of role hierarchies, a role may still have the permissions even if they have been revoked from the role. In the latter case, a user with this role is able to access objects in the permission and has operations on the objects. The problems arising in processes of assigning and revocation are evident.

Authorization granting problem – How to check whether a permission is in conflict with the permissions of a role?

Authorization revocation problem – How to find whether permissions of a role have been revoked from the role or not?

For example, Figure 2 shows a system administrative role (BankSO) in a bank to manage regular roles such as AUDITOR, TELLER, ACCOUNT_REP and MANAGER. Role MANAGER inherits AUDITOR, TELLER and ACCOUNT_REP. ACCOUNT_REP has a SSD relationship with AUDITOR as well as DSD relationship with TELLER.

The administrative role BankSO can assign audit permission or cash operation permission to a role but not both, otherwise it compromises the security of a bank system. It is easy to find conflicts between permissions when assigning permissions to a role in a small database system but it is hard to find them when there are thousands of permissions in a system. Our aim is to provide relational algebra algorithms to solve the problems and then automatically check conflicts when assigning and revoking.

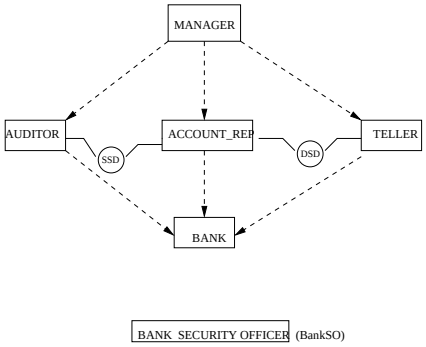

Figure 2. Administrative role and role Relationships in a bank

Some relations in set REL are detailed below.

ROLES - This relation has $(n+1)$ attributes when there are n roles.

The first attribute, RoleName is the primary key for the relation, and represents the name of a role. From the second attribute to $(n+1)th$ attribute refer to other roles, their corresponding values describe the state of conflicts with the RoleName in the relation and its domain is {-1, 0}, where '-1' means conflicting with the RoleName and '0' means not.

The ROLES relation in Figure 2 is in Table 1. The attribute TELLERC shows whether the role TELLER is conflicting with the RoleName in the relation or not. For instance, in the third tuple, a user with role TELLER has conflicts with the role AUDITOR.

RoleName	MANAC	AUDC	AUD_REPC	TELLERC
MANAGER	0	0	0	0
AUDITOR	-1	0	-1	-1
AUDITOR_REP	-1	-1	0	-1
TELLER	-1	-1	-1	0

Table 1. The relation ROLES in Figure 1

PERM - It is a relation of {PermName, Oper, Object, ConfPer }:

PermName is the primary key for the table, and is the name of the permission in the system.

Oper is the name of the operation granted. It has information about the object that the operation is granted on.

Object is the database item that can be accessed by the operation. It can be a database, a table, a view, an index or a database package.

ConfPer is a set of permissions that is in conflict with the PermName in the relation.

For example, a staff in a bank cannot have both permissions of approval and funding as well as both permissions of audit and teller. The relation of PERM can be expressed as Table 2.

PermName	Oper	Object	ConfPerm
Approval	approve	cash or check	Funding
Funding	invest	cash	Approval
Audit	audit	record	Teller
Teller	transfer	cash	Audit

Table 2. An example of the relation PERM

Roles are managed by administrative roles. Senior roles are shown at the top of the hierarchies. Senior roles inherit permissions from junior roles. Let $x > y$ denote x is senior to y with obvious extension to $x \geq y$.

SEN-JUN - This is a relation of roles in a system. Senior is the senior of the two roles. Table 3 expresses the SEN-JUN relationship in Figure 2.

Senior	Junior
MANAGER	AUDITOR
MANAGER	TELLER
MANAGER	AUDITOR_REP
TELLER	BANK
AUDITOR	BANK

Table 3. SEN-JUN table in Figure 1

ROLE-PERM - is a relationship between the ROLES and the PERM, listing what permissions are granted to what roles. It has two attributes:

RoleName is a foreign key RoleName from the table ROLES.

PermName is a foreign key PermName from the table PERM which is assigned to the role.

Suppose the permission Approval is assigned to role TELLER and the permission Funding to role MANAGER, Table 4 expresses the permission-role relationship.

RoleName	PermName
MANAGER	Funding
TELLER	Approval

Table 4. Permission-Role table

Based on these relations, we will describe how to solve the Authorization granting problem and revocation problem in the next section.

3 Authorization granting and revocation algorithms based on relational algebra

We develop granting and revocation algorithms based on relational algebra in this section. The notion of a prerequisite condition, Can-assignp and Can-revokep is a key part in the processes of permission_role assignment.

Prerequisite condition is an expression using Boolean operators $\wedge$ and $\vee$ on terms of the form x and $\bar{x}$ where x is a role and $\wedge$ means "and", $\vee$ means "or". A prerequisite condition is evaluated for a permission p by interpreting x to be true if $(\exists x' \geq x), (p, x') \in PA$ and $\bar{x}$ to be true if $(\forall x' \geq x), (p, x') \notin PA$, where PA is a set of permission-role assignments. $\diamond$

For a given set of roles R let CR denote all possible prerequisite conditions that can be formed using the roles in R. Not every administrator can assign a permission to a role. The relation of **Can-assignp** $\subseteq AR \times CR \times 2^R$ provides what permissions can be assigned by administrators with prerequisite conditions, where AR is a set of administrative roles.

For example, the meaning of *Can-assignp* (x, y, Z) is that a member of the administrative role x can assign a permission whose current membership satisfies the prerequisite condition y to be a member of roles in range Z.

Permission-role assignment (PA) is authorized by Can-assignp relation. Table 5 shows the Can-assignp relations with the prerequisite conditions in the example.

The meaning of Can-assignp (BankSO, BANK $\wedge\overline{TELLER} \wedge \overline{AUDITOR}$, [AUDITOR_REP, AUDITOR_REP]) is that a member of the administrative role BankSO can assign a permission whose current membership satisfies the prerequisite condition BANK $\wedge\overline{TELLER} \wedge \overline{AUDITOR}$ to be a member of role AUDITOR_REP. To identify a role range within the role hierarchy, the following closed and open interval notation is used.

$$[x, y] = \{r \in R | x \geq r \wedge r \geq y\}$$
$$(x, y] = \{r \in R | x > r \wedge r \geq y\}$$
$$[x, y) = \{r \in R | x \geq r \wedge r > y\}$$
$$(x, y) = \{r \in R | x > r \wedge r > y\}$$

Supposing an administrator role ADrole wants to assign a permission p_j to a role r with a set of permissions P. P^* is an extension of P, $P^* = \{p | p \in P\} \cup \{p | \exists r' \in R, r' < r, (p, r') \in PA\}$.

Admin.role	Prereq.Condition	Role Range
BankSO	BANK $\wedge TELLER \wedge \overline{AUDITOR}$	[AUDITOR_REP, AUDITOR_REP]
BankSO	BANK $\wedge \overline{TELLER} \wedge \overline{AUDITOR\_REP}$	[AUDITOR, AUDITOR]
BankSO	BANK $\wedge \overline{AUDITOR} \wedge \overline{AUDITOR\_REP}$	[TELLER, TELLER]

Table 5. Can-assignp relation in Figure 1

Authorization granting algorithm Grantp(ADrole, r, p_j)
Input: ADrole, role r and a permission p_j.
Output: true if ADrole can assign the permission p_j to r with no conflicts; false otherwise.
Begin:
Step 1. /* *Whether the ADrole can assign the permission p_j to r or not* */

Let

$$S = \pi_{Prereq.Condition}(\sigma_{admin.role=ADrole}(Can - assignp))$$

/* *S is a set of Prerequisite condition associated with ADrole* */
and

$$R = \pi_{RoleName}(\sigma_{PermName=p_j}(ROLE - PERM))$$

/* *R is a set of role associate with the permission p_j* */

if $S_1 = S \cap R \neq \phi$,
then there exists role $r_1 \in S_1$, such that
$(p_j, r_1) \in PA$ and
$r_1 \in \pi_{Prereq.Condition}(\sigma_{admin.role=ADrole}(Can - assignp)$
go to step 2 /* *p_j is satisfied the Prerequisite condition to be assigned by ADrole in Can-assign* */

else, return false and stop. /**the admini.role has no right to assign the permission p_j to role r* */

Step 2. /**whether the permission p_j is conflicting with permissions of r or not*, in other words, whether p_j is conflicting with permission set P^* or not*/
Let
$ConfPermS = \pi_{ConfPerm}(\sigma_{PermName=p_j}(PERM))$
/* *It is the conflicting permission set of the permission p_j* */
if $ConfPermS \cap P^* \neq \phi$,
then
p_j is a conflicting permission with role r, *return* false;
else
return true.
/* *p_j is not a conflicting permission with r* */ ◇

Theorem 1 The authorization granting algorithm can prevent conflicts when assigning a permission to a role.

Proof Assuming an administrator role ADrole wants to assign a permission p_j to a role which associates with a permissions set P. While step 1 in the algorithm has checked whether the ADrole can assign the permission p_j to a role or not, the second step has decided whether the permission p_j is conflicting with permissions in P^* or not. Indeed, p_j can be assigned to the role if for all $p_i \in P^*$, p_i is not in the conflicting permission set of p_j. Otherwise p_j is a conflicting permission with P^*. ◇

The authorization granting problem is solved by the authorization granting algorithm. Computing S in the first step takes time proportional to n^2 if n is presented as the number of roles. This is because computing $\sigma_{admin.role=ADrole}(Can-assignp)$ and $\pi_{Prereq.Condition}(\sigma_{admin.role=ADrole}(Can\ -\ assignp))$ needs time $O(n)$. It spends time $O(n)$ to compute R and $O(n^2)$ for S_1. Thus, the step 1 takes time $O(n^2)$. In the second step, the computations of $\sigma_{PermName=p_j}(PERM)$ and $\pi_{ConfPerm}(\sigma_{PermName=p_j}(PERM))$ spend time $O(m)$ and $O(1)$ respectively when there are m permissions in the system. It needs time $O(m^2)$ to decide whether there is a permission in $ConfPermS \cap P^*$ or not . Therefore the total time spent in the authorization granting algorithm is proportional to $(n^2 + m^2)$.

Corollary 1 The authorization granting algorithm has time complexity $O(n^2 + m^2)$ for the case of n roles and m permissions in a system.

There are related subtleties that arise in RBAC concerning the interaction between granting and revocation of permission-role membership. A relation **Can-revokep**$\subseteq AR \times 2^R$ shows which permissions in what role range can be revoked by administrative, where AR is a set of administrative roles. The meaning of Can-revokep (x, Y) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke the membership of a permission from any role $y \in Y$, where Y defines the *range of revocation*. Table 6 gives an example of the Can-revokep relation.

Admin.role	Role Range
BankSO	[Bank, MANAGER)

Table 6. An example of Can-revokep

The meaning of Can-revokep

$(BankSO, [Bank, MANAGER))$ in Table 6 is that a member of the administrative role BankSO can revoke the membership of a permission from any role in [Bank, MANAGER).

Due to role hierarchy, a role x' has all permissions of a role x when $x' > x$. A permission p is an *explicit member* of a role x if $(p, x) \in PA$, and p is an *implicit member* of role x if for some role $x' < x, (p, x') \in PA$. Weak revocation has an impact only on explicit membership. For weak revocation, the membership of a permission is revoked only if the permisiion is an explicit member of the role. Therefore, weak revocation from a role x has no effect when a permission is not an explicit member of the role x. To solve the authorization revocation problem, we need to revoke the explicit member of a role first if a permission is an explicit member, then revoke the implicit member.

Following are two algorithms for revocation of a permission p_j from a role r by an administrative role ADrole. The first one is weak revocation algorithm and another one is strong revocation algorithm. The weak revocation only revokes explicit membership from a role and does not revoke implicit membership but the strong revocation revokes both explicit and implicit members.

Weak revocation Algorithm Weak_revoke(ADrole, r, p_j)
Input: ADrole, a roles r and a permission p_j.
Output: true if ADrole can weakly revoke role p_j from r; false otherwise.
Begin:
if $p_j \notin \{p|(p, r) \in PA\}$,
return false; /* *there is no effect with the operation of the weak revocation since the permission* p_j *is not an explicit member of the role r**/
else /* p_j *is an explicit member of* r */
Let
RevokeRange
$= \pi_{RoleRange}(\sigma_{admin.role=ADrole}(Can - revokep))$
/* *The role range can be revoked by ADrole**/
and
$Roleswithp_j = \pi_{RoleName}(\sigma_{PermName=p_j}(ROLEPERM))$

/**Roles with permission* p_j**/
If $RevokeRange \cap Roleswithp_j \neq \phi$
return true; /**the* p_j*can be revoked **/
else, return, false. /* *ADrole has no right torevoke the permission* p_j *from the role **/ ◇

We have the following result with the weak revocation algorithm.

Theorem 2 A permission p_j is revoked by the weak revocation algorithm $Weak\_revoke(ADrole, r, p_j)$ if the permission is an explicit member of the role r and the ADrole has the right to revoke p_j from the Can-revokep relation. ◇

It takes time $O(m)$ to check if $p_j \notin \{p|(p, r) \in PA\}$ when there are m permissions in a system. The computations of $\sigma_{admin.role=ADrole}(Can - revokep)$ and $\pi_{RoleRange}(\sigma_{admin.role=ADrole}(Can - revokep))$ take time $O(n)$ when there are n roles in the system. The process $\pi_{RoleName}(\sigma_{PermName=p_j}(ROLEPERM))$ needs time $O(n)$. To check whether $RevokeRange \cap Roleswithp_j \neq \phi$ or not needs time $O(n^2)$. Thus, the time complexity of the Weak revocation algorithm is $O(n^2 + m)$.

Corollary 2 Weak revocation algorithm has time complexity $O(n^2 + m)$ when there are n roles and m permissions in a system.

A role still owns a permission of a system which has been weakly revoked if the role is senior to another role associated with the permission. To solve the authorization revocation problem, we need strong revocation which requires revocation of both explicit and implicit membership. Strong revocation of a permission's membership in role r requires that the permission be removed not only from explicit membership in r, but also from explicit and implicit membership in all roles junior to r. Strong revocation therefore has a cascading effect up-wards in the role hierarchy.

Strong revocation algorithm Strong_revoke(ADrole, r, p_j)
Input: ADrole, a role r and a permission p_j.
Output: true, if it can strong revoke the permission p_j from r; false otherwise.
Begin:
if $p_j \notin P^*$,
return false; /* *there is no effect of the strong revocation since the permission is not an explicit and implicit member of the role* r **/*
else,
1. *if* $p_j \in P$, do Weak_revoke(ADrole, r, p_j);
/**p_j is weakly revoked from r**/;
2. Suppose
$Jun = \pi_{Junior}(\sigma_{Senior=r}(SEN - JUN))$,
for all $y \in Jun$, do Weak_revoke(ADrole, y, p_j);
/**the permission* p_j *is weakly revoked from all such* $y \in Jun$**/.
If all the weak revocations are successful,
return true;
otherwise,
return false. /* if one weak revocation cannot finish*/ ◇

It should be noted that Strong revocation algorithm does not work if ADrole has no right to revoke p_j from any role in Jun. We have the following consequence.

Theorem 3 The explicit and implicit member of permission p_j are revoked from the role r by the Strong revocation al-

gorithm $Strong\_revokep(ADrole, r, p_j)$ if the ADrole has the right to revoke p_j from the Can-revokep relation.

Corollary 3 The authorization revocation problem is solved by the Weak revocation algorithm and Strong revocation algorithm.

It needs $O(m)$ to check whether $p_j \notin P^*$ if there are m roles in a system. The computations of $\sigma_{Senior=r}(SEN - JUN)$ and $\pi_{Junior}(\sigma_{Senior=r}(SEN - JUN))$ take time proportional to l ($l < n$) where l is the number of tuples in the relation SEN-JUN and n is the number of roles in the system. It takes time proportional to $(n^2 + m)$ to do a weak revocation. A role may be junior $(n-1)$ roles, hence all weak revocations need time $O(n * (n^2 + m))$. Other operations $p_j \in P$ and $y \in Jun$ takes time $O(m)$ and $O(n)$ respectively. Therefore the total time spent with the Strong revocation algorithm is $O(n * (n^2 + m))$.

Corollary 4 The Strong revocation algorithm has time complexity $O(n * (n^2 + m))$ when there are n roles and m permissions in a system.

In the remaining parts of this paper, the new relational algebra approaches will be used with a payment scheme. We review the payment scheme first.

4 Review of the anonymity scalable electronic payment scheme

We have developed an anonymity self-scalable payment scheme [20]. The payment scheme provides different degrees of anonymity for consumers. Consumers can decide the levels of anonymity. They can have a low level of anonymity if they want to spend coins directly after withdrawing them from the bank. Consumers can achieve a high level of anonymity through an anonymity provider (AP) agent without revealing their private information and are secure in relation to the bank because the new certificate of a coin comes from the AP agent who is not involved in the payment process. The scheme is briefly reviewed below.

Electronic cash has sparked wide interest among cryptographers ([14, 23, 12, 19], etc.). In its simplest form, an e-cash system consists of three parts (a bank, a consumer and a shop) and three main procedures as shown in Figure 3 (Withdrawal, Payment and Deposit). In a coin's life-cycle, the consumer first performs an account establishment protocol to open an account with the bank.

The consumers and the shops maintain an account with the bank, while:

1. A consumer withdraws electronic coins from his/her account, by performing a withdrawal protocol with the bank over an authenticated channel.

2. The consumer spends a coin by participating in a payment protocol with a shop over an anonymous channel.

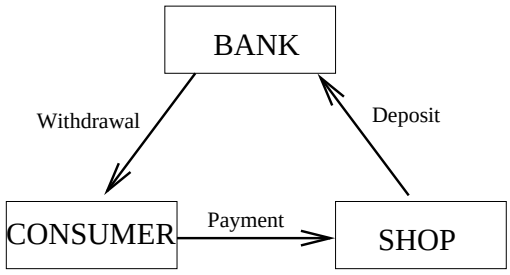

Figure 3. Basic processes of an electronic cash system

3. The shop performs a deposit protocol with the bank, to deposit the consumer's coin into its account.

There are also three additional processes: the bank setup, the shop setup, and the consumer setup (account opening). They describe the system initialization, namely the creation and posting of public keys and opening of bank accounts. Although they are certainly parts of a complete system, these are often omitted as their functionalities can be easily inferred from the description of the three main procedures.

Besides the basic participants, a third party named anonymity provider (AP) agent is involved in the scheme. The AP agent will help the consumer to get the required anonymity but will not be involved in the purchase process. The model is shown in Figure 4. The AP agent gives a certificate to the consumer when s/he needs a high level of anonymity.

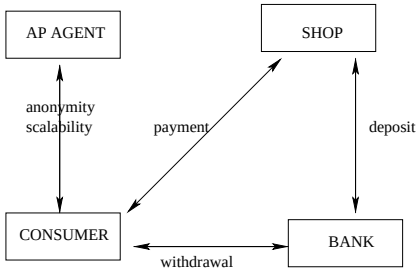

Figure 4. Electronic cash model

The scheme in [20] includes two basic processes in system initialization (bank setup and consumer setup) and three main protocols: a withdrawal protocol with which a consumer withdraws electronic coins from a bank while his account is debited; a payment protocol with which the consumer pays the coin to a shop; and a deposit protocol with which the shop deposits the coin in the bank and has its account credited. If a consumer wants to get a high level of anonymity after getting a coin from the bank (withdrawal), s/he can contact the AP agent.

From the viewpoint of banks, consumers can improve anonymity if they are worried about disclosure of their identities. This is a practical payment scheme for Internet purchases because it has provided a solution with different anonymity requirements for consumers. The security of the scheme has been discussed in [20]. We will analyze its management with the relational algebra algorithms.

5 Applications of the relational algebra algorithms

The new relational algebra algorithms will be applied to the payment scheme in this section. We add a manager role (M1) etc in an AP agent, a manager role (M2) etc in a bank, a manager role (M3) etc in a shop and some administrative roles Senior Officer(SSO) etc in the system as shown in Figure 5 and Figure 6. A hierarchy of roles and a hierarchy of administrative roles are also shown in these two Figures. Senior roles are shown towards the top of the hierarchies and junior are to the bottom. Senior roles inherit permissions from junior roles. Permissions can be granted to or revoked from the roles in Figure 5 by the administrative roles in Figure 6.

5.1 An application of the authorization granting algorithm

Figure 5 shows that role E is the most junior to all other employees in the new system and role Director (DIR) is the most senior to all other employees. Figure 6 shows the administrative role hierarchy which co-exist with the roles in Figure 5. The senior-most role is the Senior Security Officer (SSO). Our interest is in the administrative roles junior to SSO. These consist of three security officer roles (APSO, BankSO and ShopSO) with the relationships illustrated in the Figure 6. Table 2 and Table 7 show part of the relations between permissions and between permissions and roles in the scheme.

RoleName	PermName
Director (DIR)	Funding
Director (DIR)	Approval
Director (DIR)	Teller
TELLER	Approval
FPS	Approval
Bank	Teller

Table 7. ROLE-PERM table in the scheme

Based on the role hierarchy in Figure 5 and administrative role hierarchy in Figure 6, we define the Can-assignp relation shown in Table 8.

Admin.role	Prereq.Condition	Role Range
NSSO	DIR	[M1, M1]
NSSO	DIR	[M2, M2]
NSSO	DIR	[M3, M3]
APSO	FPS $\wedge\overline{OP}$	[QC, QC]
APSO	FPS $\wedge\overline{QC}$	[OP, OP]
BankSO	FPS $\wedge\overline{TE} \wedge \overline{AU}$	[AC, AC]
BankSO	FPS $\wedge\overline{TE} \wedge \overline{AC}$	[AU, AU]
BankSO	FPS $\wedge\overline{AU} \wedge \overline{AC}$	[TE, TE]
ShopSO	FPS $\wedge\overline{SALER}$	[AUDITOR, AUDITOR]
ShopSO	FPS $\wedge\overline{AUDITOR}$	[SALER, SALER]

Table 8. Can-assignp

Let us consider the NSSO tuples in Table 8 (the analysis for APSO, BankSO and ShopSO are similar). The first tuple authorizes NSSO to assign permissions with the prerequisite condition role DIR into members of M1 in the AP agent (AP). The second and the third one authorize NSSO to assign permissions with the prerequisite condition DIR to be a member of M2 and M3 respectively. Similarly, the fourth tuple authorizes APSO to assign permissions with the prerequisite condition $FPS \wedge \overline{OP}$ to be members of operators (QC). The fourth and fifth tuple show that the APSO can grant a permission who is a member of the AP agent into one but not both of QC and OP. This illustrates how mutually exclusive roles can be forced by permission-role assignment.

Assume the role FPS with permission set $P = \{Approval\}$ and $P^* = P = \{Approval\}$. The administrative role NSSO wants to assign the permission *Teller* to the role FPS. Using the granting algorithm *Grantp(NSSO, FPS, Teller)*, the first step,

$$\begin{aligned} S &= \pi_{Prereq.Condition}(\sigma_{Admin.role=NSSO}(Can-assignp)) \\ &= \{DIR\} \end{aligned}$$

and

$$\begin{aligned} R &= \pi_{RoleName}(\sigma_{PermName=Teller}(ROLE-PERM)) \\ &= \{DIR, Teller\} \end{aligned}$$

Since $R \cap S = \{DIR\} \neq \phi$. This means NSSO can assign permission *Teller* to role FPS.

The second step, based on Table 2,

$$\begin{aligned} Conf-perm &= \pi_{ConfPerm}(\sigma_{PermName=Approval}(PERM)) \\ &= \{Funding\} \end{aligned}$$

and

$$Conf-perm \cap P^* = \phi.$$

It means no conflicts when assigning the permission Teller to role FPS.

5.2 Application of the authorization revocation algorithm

Table 9 and Table 3 give the Can-revokep and a part of senior-junior relationship of the payment scheme.

Admin.role	Role Range
NSSO	[FPS, DIR)
APSO	[AP, M1)
BankSO	[Bank, M2)
ShopSO	[Shop, M3)

Table 9. Can-revokep

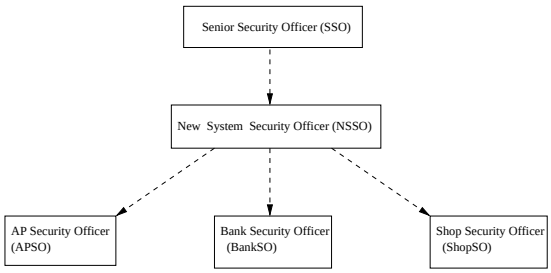

Figure 6. Administrative role assignment in the scheme

Based on the Table 7, The permission *Approval* is an explicit member of role DIR, TELLER and FPS in the scheme. If Alice, with the activated administrative role BankSO, weakly revoke *Approval*'s membership from TELLER, the revocation is successful by the weak revocation algorithm Weak_revoke(BankSO, TELLER, Approval). This is because

$$RevokeRange \cap RoleswithApproval$$
$$= \{TELLER\}$$
$$\neq \phi$$

where

$$Revoke - Range$$
$$= \pi_{RoleRange}(\sigma_{Admin.role=APSO}(Can - revokep))$$
$$= [Bank, M2)$$

and

$$RoleswithApproval$$
$$= \pi_{RoleName}(\sigma_{PermName=Approval}(ROLEPERM))$$
$$= \{DIR, TELLER, FPS\}$$

Approval continues to be an implicit member of TELLER since FPS is junior to TELLER and *Approval* is an explicit member of FPS. It is necessary to note that Alice should have enough power in the session to weakly revoke *Approval* from explicitly assigned roles. For instance, if Alice has activated BankSO and then tries to weakly revoke *Approval* from FPS, she is not allowed to proceed because BankSO does not have the authority of weak revocation from FPS according to the Can-revokep relation in Table 9. Therefore, if Alice wants to revoke *Approval*'s explicit membership as well as implicit membership from TELLER by weak revocation, she needs to activate NSSO and weakly revoke *Approval* from TELLER and FPS.

If Alice, with the activated administrative role NSSO, strongly revokes *Approval*'s membership from TELLER, then *Approval* is removed not only from explicit membership in TELLER, but also from explicit (and implicit) membership in all roles junior to TELLER. Actually, using the strong revocation algorithm Strong_revoke(NSSO, TELLER, Approval), $P = \{Approval\} = P^*$. It does need to do Weak_revoke (NSSO, TELLER, Approval) since $Approval \in P$. The junior set of role TELLER is {FPS}. Then the permission *Approval* has been removed from FPS as well as TELLER by running Weak_revoke (NSSO, TELLER, Approval) and Weak_revoke (NSSO, FPS, Approval). However, *Approval* still has a membership of DIR since it is not junior roles to TELLER based on the role hierarchy of Figure 5. This brings about the same result as weak revocation from TELLER and FPS by NSSO. Note that all implied revocations downward in the role hierarchy should be within the revocation range of the administrative roles that are active in a session. For instance, if Alice activates BankSO and tries to strongly revoke *Approval* from TELLER, she is not allowed to proceed because FPS is junior to TELLER but it is out of the BankSO's Can-revokep range in Table 9.

6 Related work

There are several other related works on role-based access control models [1], an oracle implementation for permission-role assignment [17] and relational databases [13].

A role-based separation of duty language (RSL 99) has been recently proposed [1]. It has given a formal syntax and semantics for RSL99 and has demonstrated its soundness and completeness by using functions on conflicting permission sets. The proposal is different from ours in two aspects. First, it does not consider the case of the management for conflicting permissions. Therefore, there is no support to deal administrative roles with permissions in the proposal. By contrast, our algorithms provide a rich variety of options that can deal the document of administrative roles with permissions. Second, the algorithm RSL99 does not provide access control models. It only gives separation of duty (SOD) policies. By contrast, we present a number of specialized authorization algorithms for access control which allow administrators to authorize a permission to role or revoke a permission from roles.

The interaction between RBAC and relational databases are presented in [13]. Two experiments are described. One

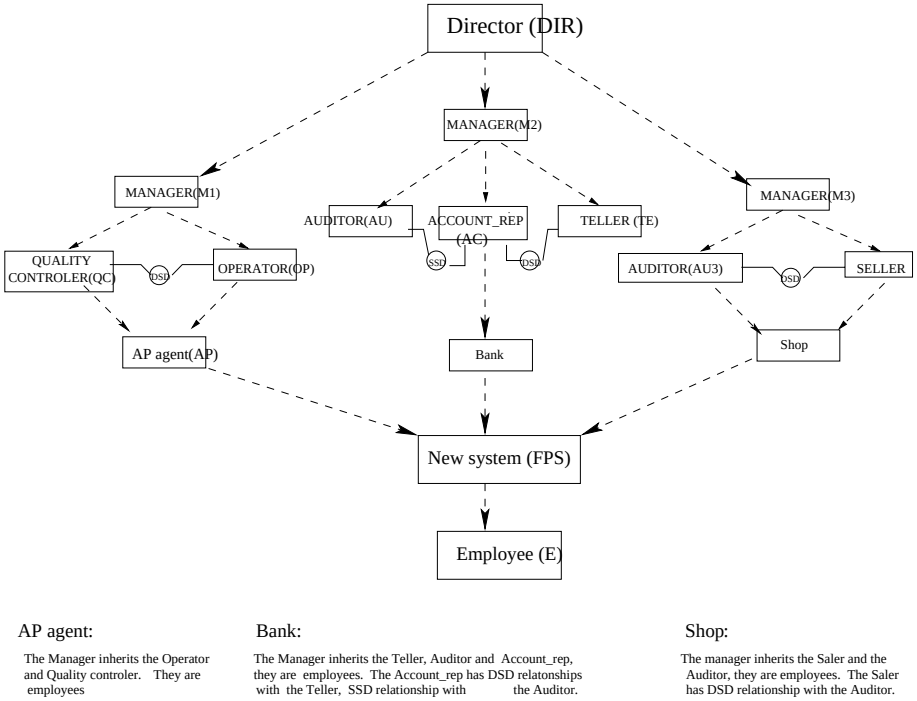

Figure 5. User-role assignment in the payment scheme

is a role-based front end to a relational database with discretionary access control. Another one is a role graph to show roles and permissions in a standard relational databases. Some relational concepts like roles, users and permissions etc are provided. Our model also support such concepts even though it has a large variety. However, the main difference between our algorithms and the scheme in [13] is, we focus on the solutions of the conflicts of permissions and the latter focuses on the correlation of RBAC with discretionary access controls. Their work discusses the relationship between roles, permissions and discretionary access controls, they did not address how to allocate permissions to roles without conflicts. In our work, we developed detailed algorithms for allocating permissions and checking the conflicts.

Finally, an oracle implementation for permission-role assignment has been proposed in [17]. In such a model, the difference between permission-role assignment (PRA97) and Oracle database management system was analysed. Furthermore, through prerequisite conditions, the paper has demonstrate how to use Oracle stored procedures to implement it. However, our work substantially differs from that proposal. Differences are due to the consistency problem which may arise in [17, 18]:

It is very difficult to keep the consistency by reflecting security requirements between global network objects and local network objects if there are hundreds of roles and thousands of permissions in a system.

This problem has been solved in our algorithms because the algorithms focus on the conflicts between permissions. The authorization granting algorithm is used to find conflicts and prevent some secret information to be derived while the strong revocation algorithm is used to check whether a role still has permissions of another role.

7 Conclusions

This paper has provided new authorization allocation algorithms for permission-role assignments that are based on relational algebra operations. They are the authorization granting algorithm, weak revocation algorithm and strong revocation algorithm. The algorithms can automatically check conflicts when granting more than one permission to a role in a system. They can prevent users associate with roles from accessing unauthorized use of facilities when the permissions of the roles are changed within the organization and demand the modification of security rights. The permissions can be allocated without compromising the security in RBAC and provide secure management for systems. The complexities of the algorithms are also analyzed. Furthermore, we have reviewed the consumer scalable anonymity payment scheme and discussed how to use the algorithms for the electronic payment scheme.

References

[1] Ahn G.J. and Sandhu R. The RSL99 Language for Role-Based Separation of Duty Constraints. In *4th ACM Workshop on Role-Based Access Control*, pages 43–54. Fairfax, VA, October, 1999.

[2] Barkley J. F. Application engineering in health care. In *Second Annual CHIN*. http://hissa.ncsl.nist.gov/rbac/proj/paper/paper.html, 1995.

[3] Barkley J. F., Beznosov K. and Uppal J. Supporting relationships in access control using role based access control. In *Third ACM Workshop on RoleBased Access Control*, pages 55–65, October, 1999.

[4] Bertino E., Castano S., Ferrari E. and Mesiti M. . Specifying and enforcing access control policies for XML document sources. *World Wide Web*, *3*, pages 139–151, Baltzer Science Publishers BV, 2000.

[5] David F.F., Dennis M.G. and Nickilyn L. An examination of federal and commercial access control policy needs. In *NIST NCSC National Computer Security Conference*, pages 107–116. Baltimore, MD, September, 1993.

[6] Feinstein H. L. Final report: Nist small business innovative research (sbir) grant: role based access control: phase 1. technical report. In *SETA Corp.*, Jan. 1995.

[7] Ferraiolo D., Cugini J. and Kuhn R. Role-based access control (rbac): Features and motivations. In *The 11th Annual Computer Security Applications Conference*, pages 241–48, New Orleans, LA, December 11- 15, 1995.

[8] Ferraiolo D. F. and Kuhn D. R. Role based access control. In *15th National Computer Security Conference*, pages 554–563, 1992.

[9] Ferraiolo D. F., Barkley J. F. and Kuhn D. R. Role-based access control model and reference implementation within a corporate intranet. In *TISSEC*, volume 2, pages 34–64, 1999.

[10] Goldschlag D., Reed M., and Syverson P. Onion routing for anonymous and private Internet connections. *Communications of the ACM*, 24(2):39–41, 1999.

[11] Lupu E., Marriott D., Sloman M. and Yialelis N. A policy based role framework for access control. In *ACM/NIST Workshop on Role-Based Access Contro*. http://www-dse.doc.ic.ac.uk/ ecl1/papers/rbac95/rbac95.pdf, 1995.

[12] Okamoto T. An efficient divisible electronic cash scheme. In *Advances in Cryptology–Crypto'95*, volume 963 of *Lectures Notes in Computer Science*, pages 438–451. Springer-Verlag, 1995.

[13] Osborn S.L., Reid L.K. and Wesson G.J. On the Interaction Between Role-Based Access Control and Relational Databases. In *IFIP WG11.3 Tenth Annual Working Conference on Database Security*, pages 139–151, July, 1996.

[14] Rivest R. T. The MD5 message digest algorithm. *Internet RFC 1321*, April 1992.

[15] Sandhu R. Role-Based Access Control. *Advances in Computers*, 46, 1998.

[16] Sandhu R. Role activation hierarchies. In *Third ACM Workshop on RoleBased Access Control*, October, 1998.

[17] Sandhu R. and Bhamidipati V. An oracle implementation of the pra97 model for permission-role assignment. In *ACM Workshop on Role-Based Access Control*, pages 13–21, 1998.

[18] Sandhu R. and Park J.S. Decentralized User-Role Assignment for Web-based Intranets. In *3th ACM Workshop on Role-Based Access Control*. Fairfax, Virginia, October, 1998.

[19] Wang H. and Zhang Y. Untraceable off-line electronic cash flow in e-commerce. In *Proceedings of the 24th Australian Computer Science Conference ACSC2001*, pages 191–198, GoldCoast, Australia, 2001. IEEE computer society.

[20] Wang H., Cao J. and Kambayashi Y. Building a consumer anonymity scalable payment protocol for the internet purchases. In *12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems*, San Jose, USA, Feb. 25-26, 2002.

[21] Wang H., Cao J. and Zhang Y. A consumer anonymity scalable payment scheme with role based access control. In *2nd International Conference on Web Information Systems Engineering*, pages 384–389, Kyoto, Japan, Dec. 3-6, 2001.

[22] Wang H., Cao J. and Zhang Y. Ticket-based service access scheme for mobile users. In *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Monash University, Melbourne, Victoria, Australia, Jan. 28-Feb. 2.

[23] Yiannis T. Fair off-line cash made easy. In *Advances in Cryptology–Asiacrypt'98*, volume 1346 of *Lectures Notes in Computer Science*, pages 240–252. Springer-Verlag, 1998.